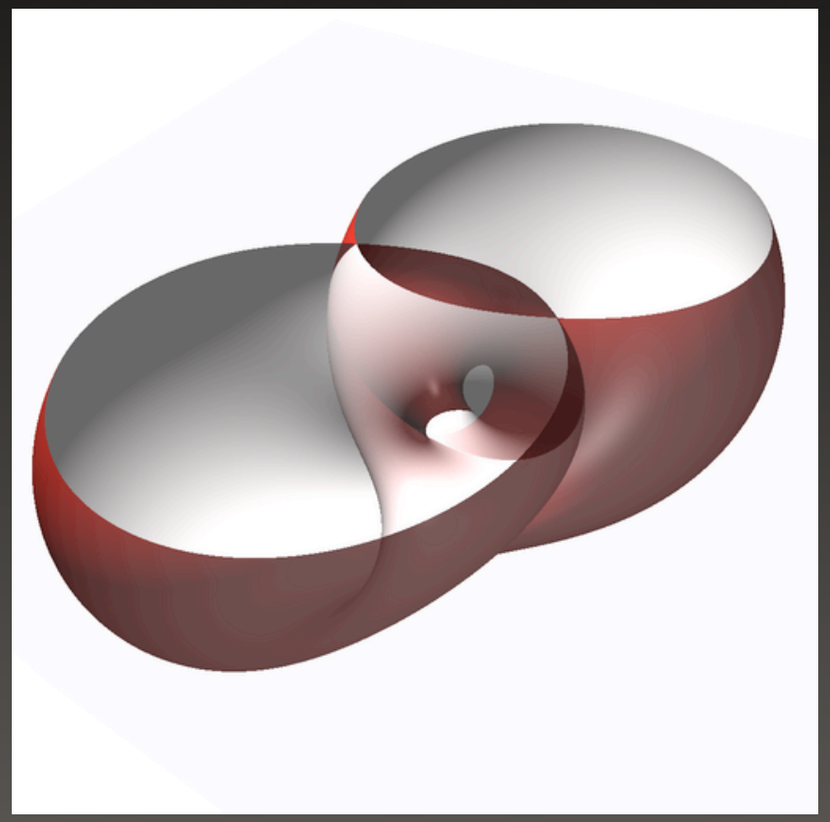
Fast and Robust Ray Tracing of Arbitrary Implicits

Aaron Knoll

co-authors:
Younis Hijazi, Ingo Wald, Andrew Kensler, Mathias Schott,
Charles Hansen, and Hans Hagen

SCI Institute, University of Utah
IRTG, University of Kaiserslautern
DOE VACET

The Goal



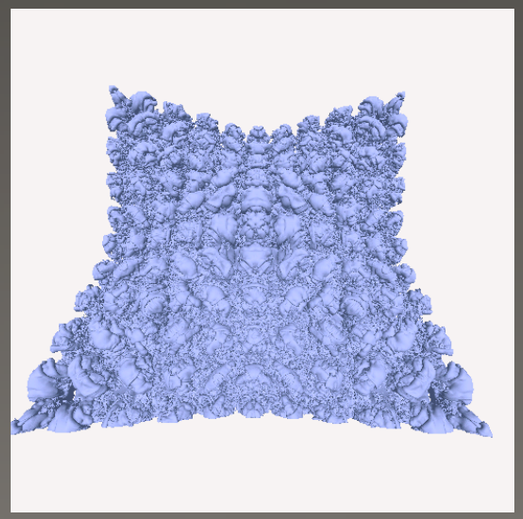
Klein bottle

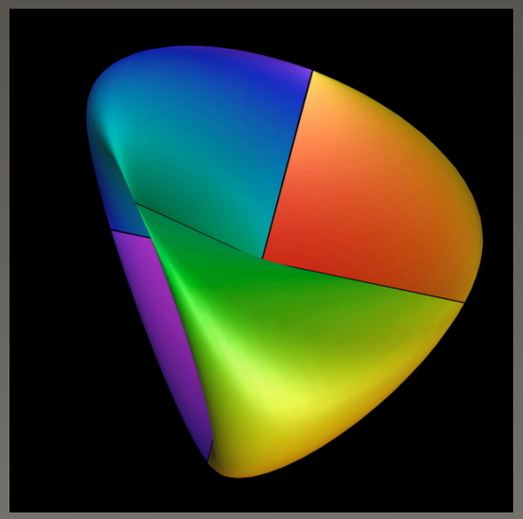
$$(x^2+y^2+z^2+2y-1)((x^2+y^2+z^2-2y-1)^2-8z^2)+16xz(x^2+y^2+z^2-2y-1)=0$$

Given **any** implicit function, render it correctly and interactively.

Motivation

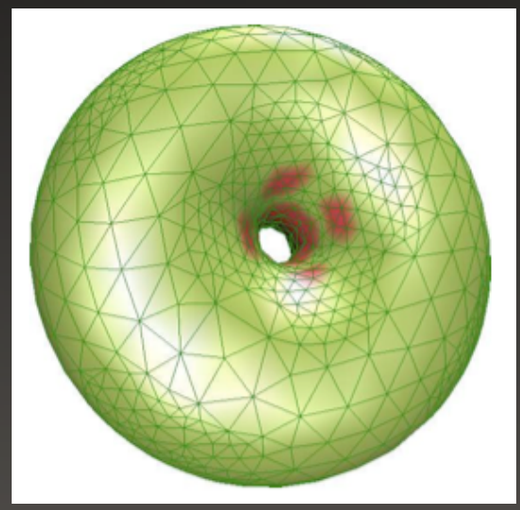
- Implicits are the *lingua franca* of spatial data reconstruction
  - filter kernels for structured volumes
  - interpolating, fitting in point clouds
- In mathematics, of visual interest in their own right
- Disadvantages:
  - many different formulations
  - direct rendering is difficult and limited
  - unwieldy, unpopular in CAD

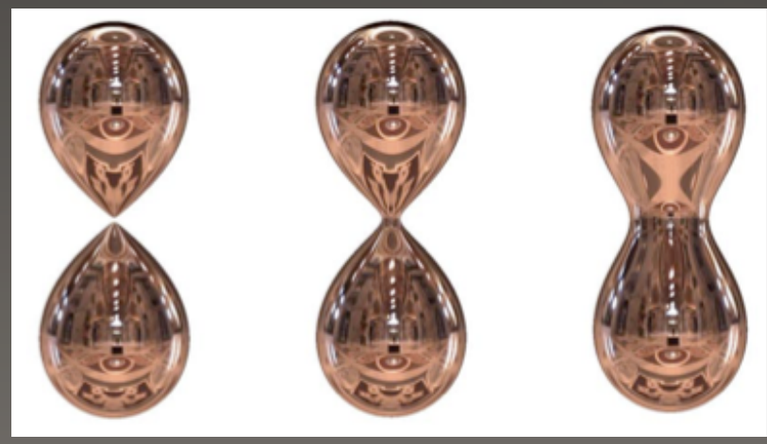




Rendering Implicits

- proxy methods
  - mesh extraction
    - marching cubes/ Bloomenthal
    - Schreiner06, Varadhan06, Paiva05
  - voxelization, volume rendering, particles, etc.
- ray tracing
  - Kalra & Barr 89
  - Mitchell 90 (interval arithmetic methods)
  - Hart 96 (signed distance functions)
  - Loop & Blinn 06, Bezier approx. on GPU





Basic Approach

- To ray trace an implicit, we have

$$f(\vec{P}) = f(x, y, z) = 0$$

$$\vec{P}(t) = \vec{O} + t\vec{D}$$

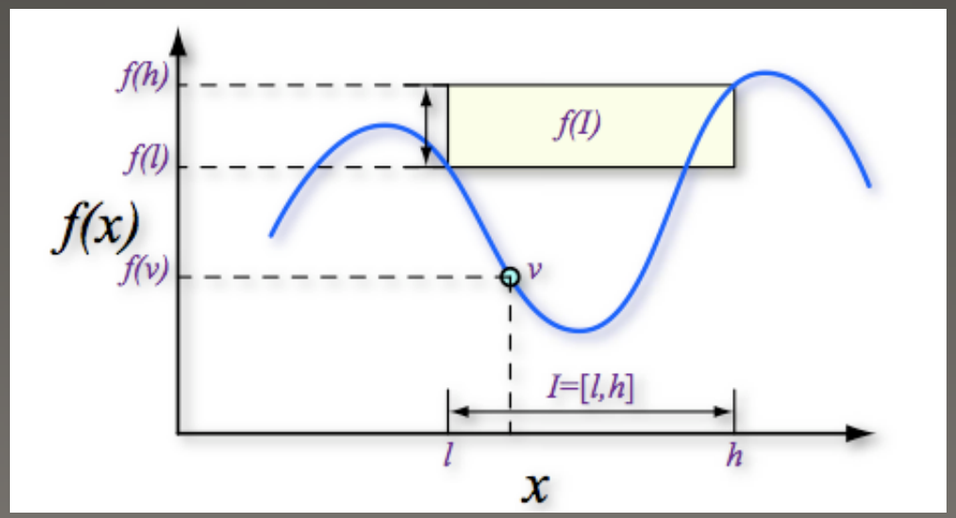
- Substituting we need to solve for *t* in:

$$f(O_x + tD_x, O_y + tD_y, O_z + tD_z) = f_t(t) = 0$$

- Ray tracing implicits is a 1D root-finding problem.

Finding roots of an arbitrary function

- Simple functions (plane, torus, conics) are trivial
  - closed-form solution
- What about order 5+, non-polynomial, or non-algebraics?
- iterative numerical methods require monotonicity
- Need to know if an interval *I* on the ray contains a root.
  - Point sampling, Descartes' Rule of Signs does not work.



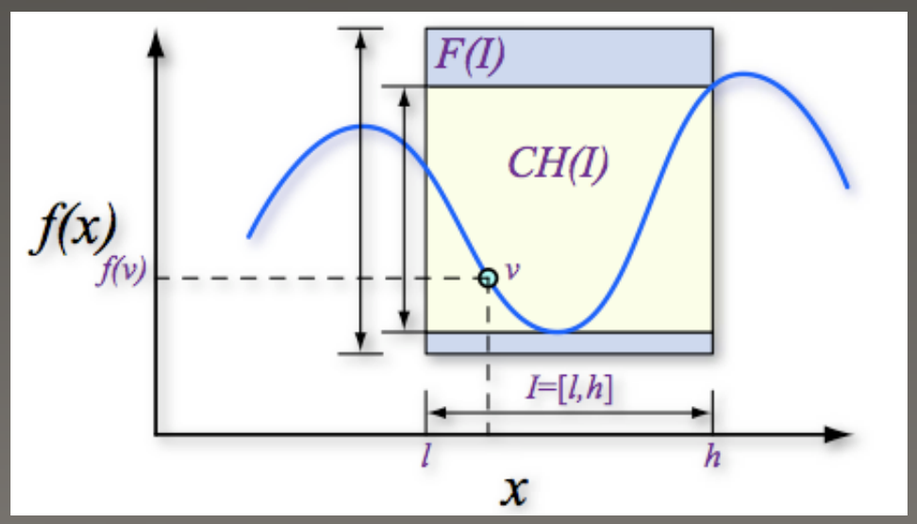
Inclusion algebra

- A composable arithmetic of operators for interval sets
  $X = [x_{lo}, x_{hi}]$

  that obeys the *inclusion property*:
  $F(x) \supseteq f(x) = f(x)|x \in X$

- Substituting these operators, we create the inclusion *extension F* of a function *f*.

- For ray tracing, this provides a spatial rejection test:
  $0 \notin F(X) \Rightarrow 0 \notin f(X)$



Interval Arithmetic

- Operations are defined on the minimum, maximum:

$$\overline{x} = [\underline{x}, \overline{x}]$$
$$\overline{x} + \overline{y} = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$$
$$\overline{x} - \overline{y} = [\underline{x} - \overline{y}, \overline{x} - \underline{y}]$$
$$\overline{x} \times \overline{y} = [min(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}, max(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}]$$

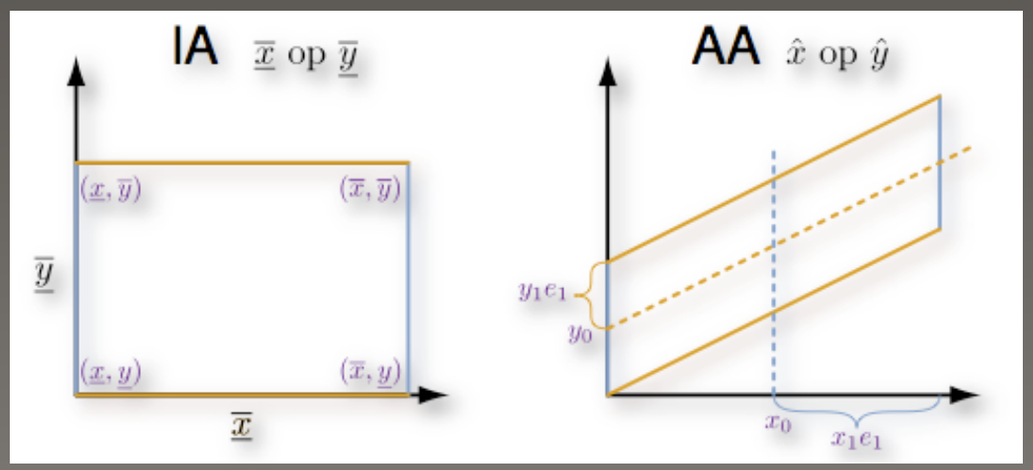
- Also called the *natural inclusion*.
- High bound overestimation from multiplication
  - || *F(X)* || can be is much larger than || *f(x)* ||
  - can be inefficient

Affine Arithmetic

- Piecewise 1st-order version of IA.
- Operates on *affine forms*:

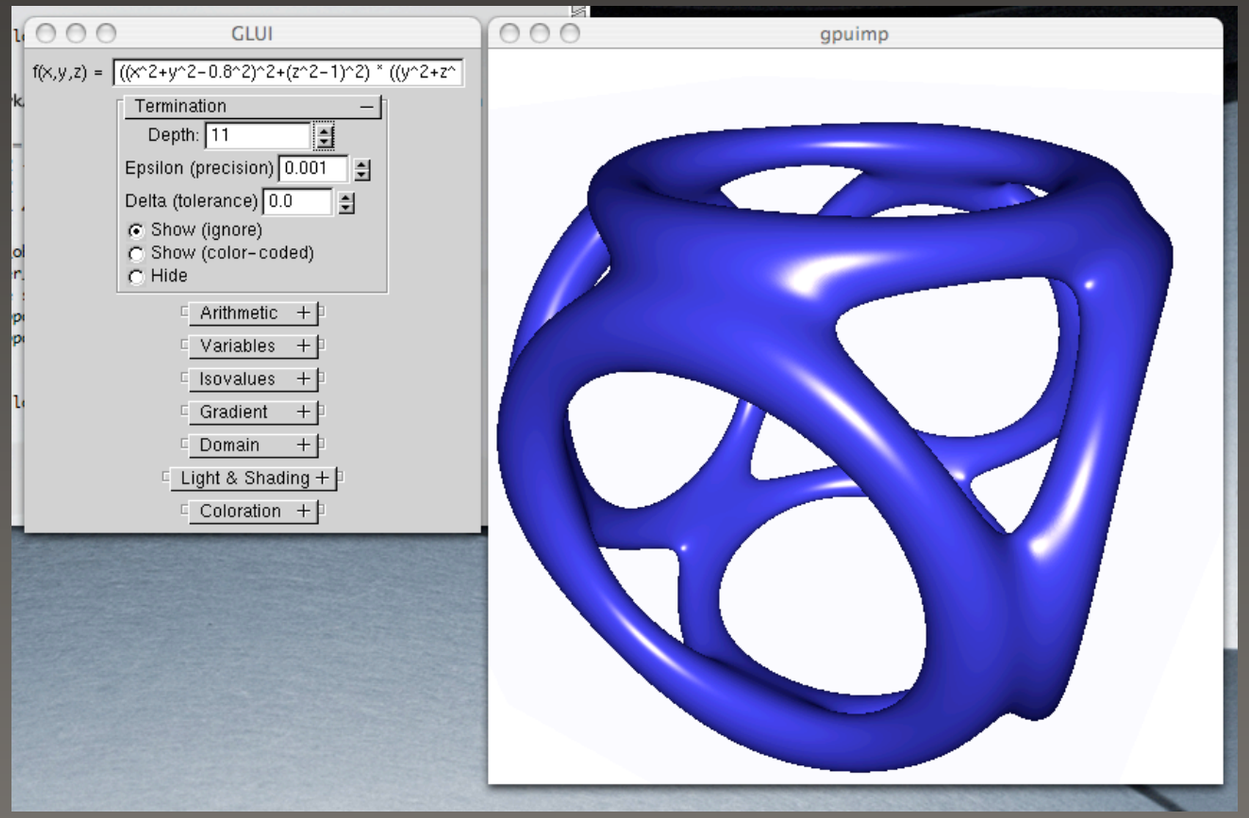
$$\hat{x} = x_0 + \sum_{i=1}^{n} x_i e_i$$

- Improves bound overestimation problem of IA
  - particularly, fixes IA multiplication.
- Reduced affine arithmetic: truncate to 3-4 terms
  - more efficient, suitable for SSE or GPU

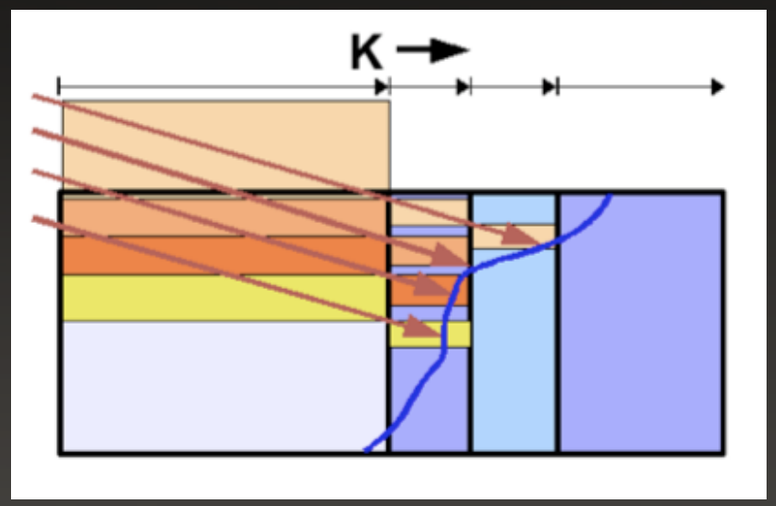


Application

- User types in any function
- Parser generates IA or AA extension of that function
- Ray tracer traverses the implicit, uses that extension as a rejection test

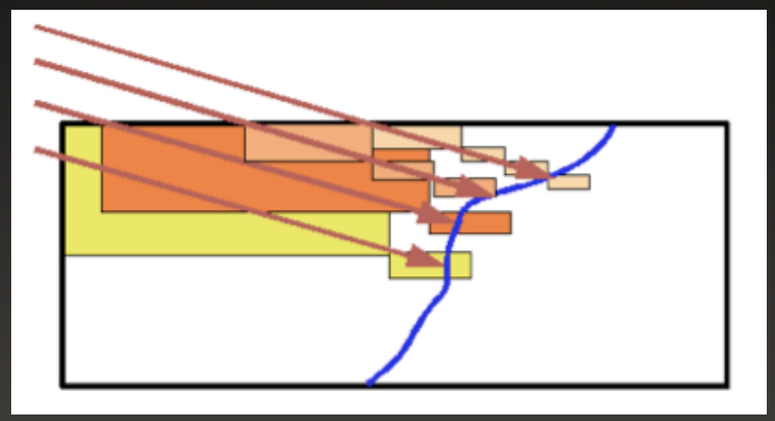


CPU coherent SSE traversal



- RT07 paper
- Since we compute the IA extension for 4 rays at a time, we want to perform interval bisection 4 at a time.
- Ideally, rays in the same packet should exhibit similar behavior during traversal
- Bisect along primary ray direction (K = an X, Y or Z axis) instead of bisecting *t*
  - fewer traversal steps, less computation
  - faster

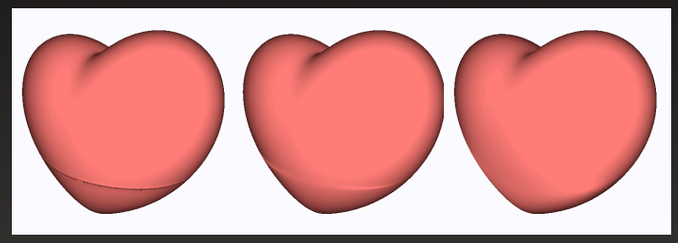
GPU Ray-Implicit Traversal

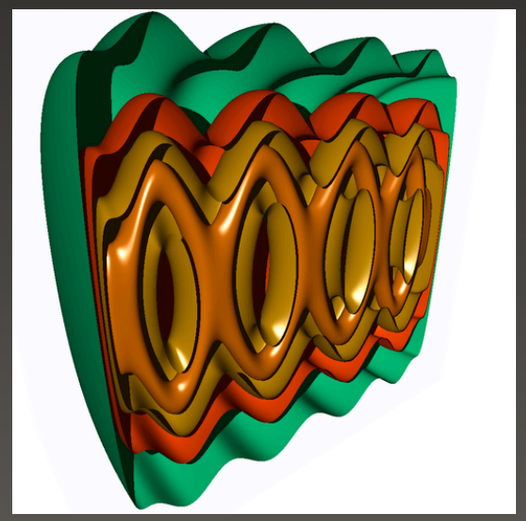


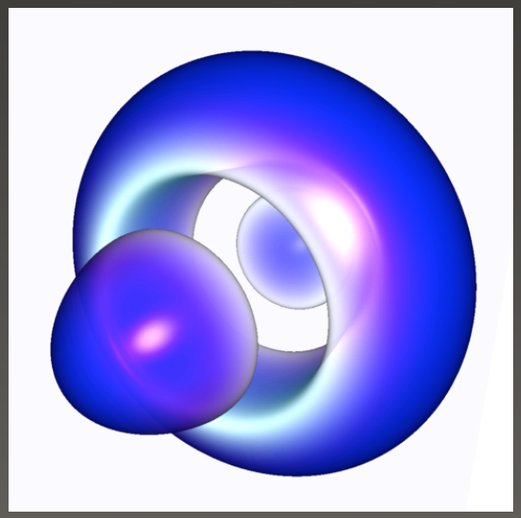
- 1 ray = 1 fragment on GPU
  - bisect *t*
  - no need to explicitly preserve coherence
- fake recursion (similar to CPU algorithm)
  - use DDA and floating point modulus
  - no per-fragment stack (inefficient on GPU)
- up to 20x faster than 4-core CPU
- Needs an NVIDIA G80
  - still could be more efficient?

Shading with Ray Tracing

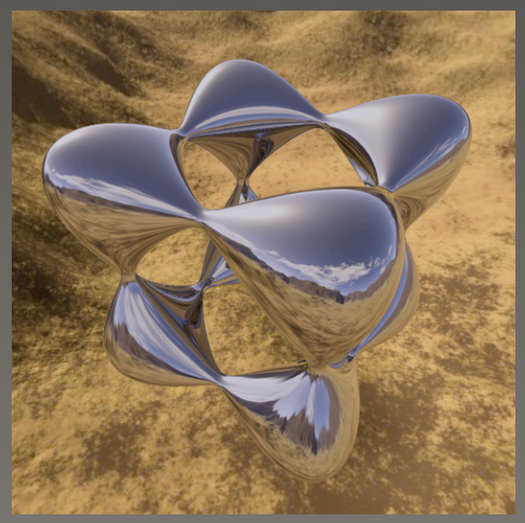
- Gradient
  - evaluate partials of implicit $\frac{df}{dx}, \frac{df}{dy}, \frac{df}{dz}$
  - use central differences





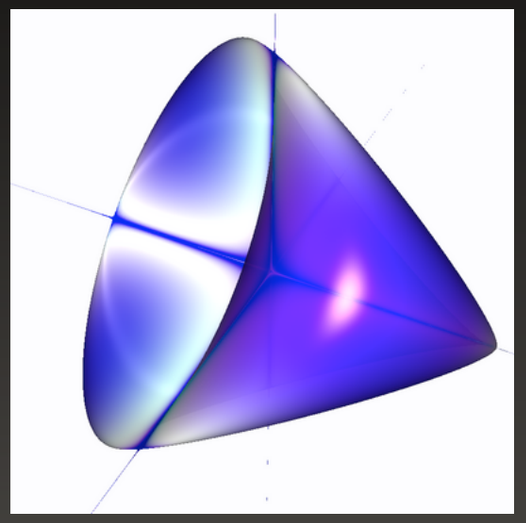


- Shadows
  - easy in RT
  - cast at lower precision
- Transparency / depth peeling
- Multi-bounce reflections
- Not significantly more expensive than ray casting.

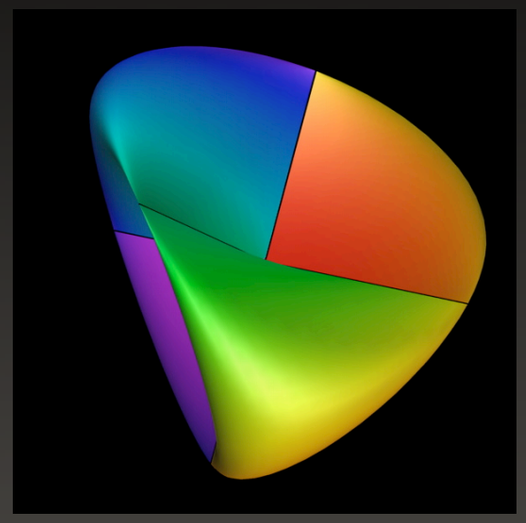


Thin feature reproduction

steiner

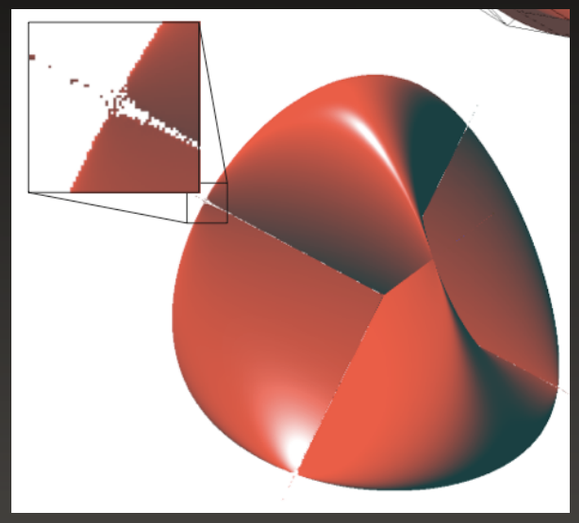


our method



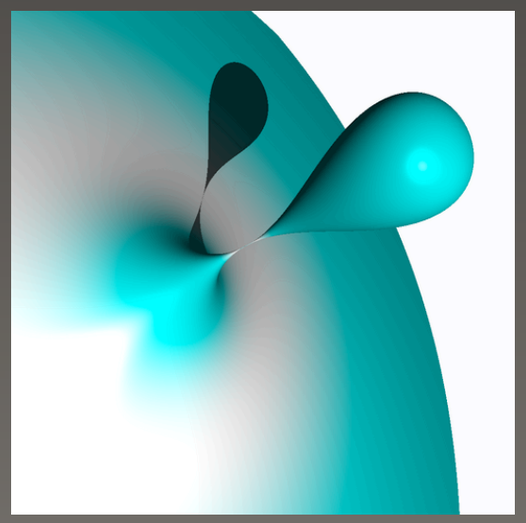
polygonization +
real-time rasterization

Bloomenthal

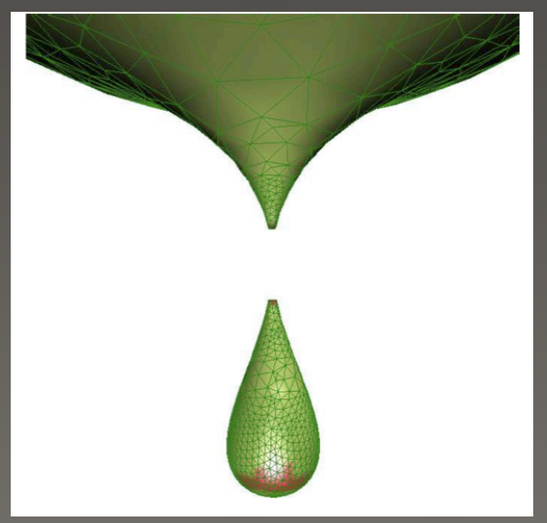


Loop & Blinn 06

teardrop

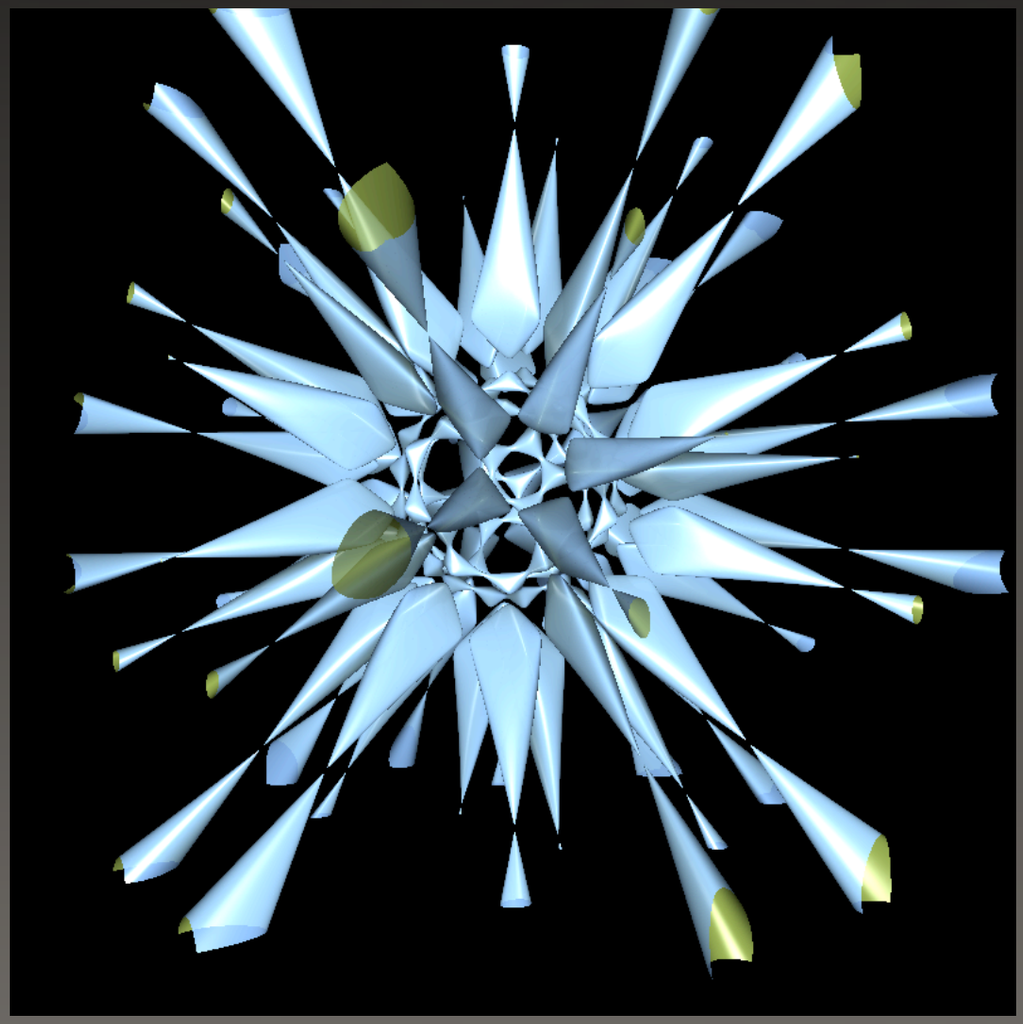


our method

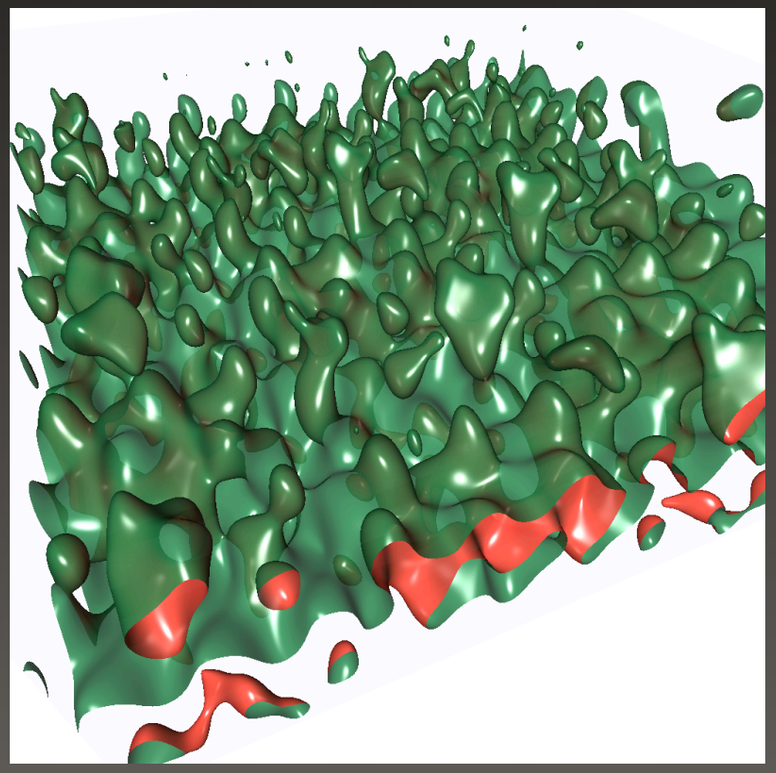


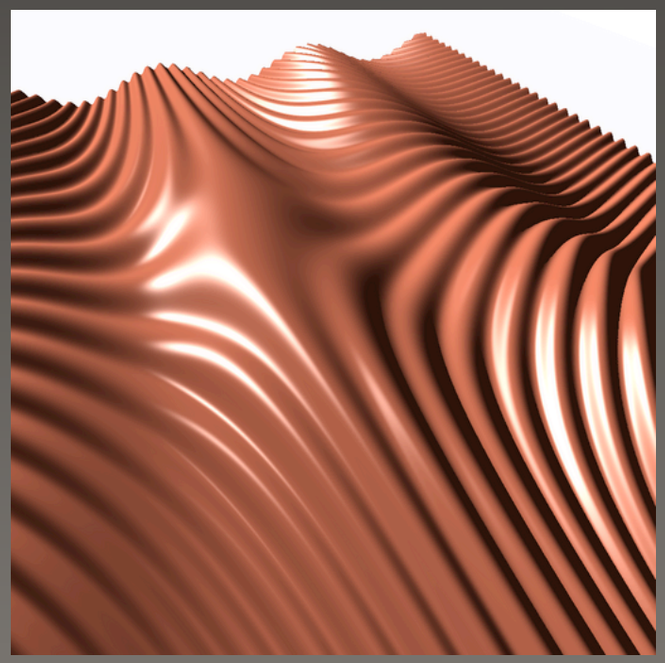
Paiva et al. 06

Mathematical visualization



Procedural geometry





Future work

- Vis applications
  - Arbitrary and higher-order reconstructions of point and volume data
  - Level set / segmentation
  - Flow vis, stream surfaces
- Graphics applications
  - procedural geometry
  - variational surface smoothing
- Even faster rendering

Thanks!

- DOE VACET
- DFG
- John C. Hart
- Thomas Lewiner
- Bill Mark
- Warren Hunt